School of Data and Computer Science, Sun Yat-sen University

E11 Decision Tree (C++/Python)

16337102 Zilin Huang

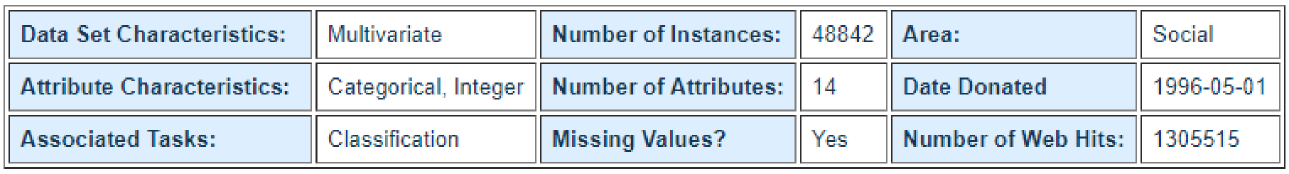
November 22, 2018

Contents

1 Datasets

The UCI dataset (`http://archive.ics.uci.edu/ml/index.php`) is the most widely used dataset for machine learning. If you are interested in other datasets in other areas, you can refer to `https://www.zhihu.com/question/63383992/answer/222718972`.

Today's experiment is conducted with the **Adult Data Set** which can be found in `http://archive.ics.uci.edu/ml/datasets/Adult`.



Data Set Characteristics:	Multivariate	**Number of Instances:**	48842	**Area:**	Social
Attribute Characteristics:	Categorical, Integer	**Number of Attributes:**	14	**Date Donated**	1996-05-01
Associated Tasks:	Classification	**Missing Values?**	Yes	**Number of Web Hits:**	1305515

You can also find 3 related files in the current folder, `adult.name` is the description of **Adult Data Set**, `adult.data` is the training set, and `adult.test` is the testing set. There are 14 attributes in this dataset:

```
>50K, <=50K.


1. age: continuous.
2. workclass: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov,
State-gov, Without-pay, Never-worked.
3. fnlwgt: continuous.
4. education: Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm,
Assoc-voc, 9th, 7th-8th, 12th, Masters, 5. 1st-4th, 10th, Doctorate, 5th-6th,
Preschool.
5. education-num: continuous.
6. marital-status: Married-civ-spouse, Divorced, Never-married, Separated,
Widowed, Married-spouse-absent, Married-AF-spouse.
7. occupation: Tech-support, Craft-repair, Other-service, Sales,
Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct,
Adm-clerical,Farming-fishing,Transport-moving,Priv-house-serv,Protective-serv,
Armed-Forces.
8. relationship: Wife,Own-child,Husband,Not-in-family,Other-relative,Unmarried.
9. race: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.
10. sex: Female, Male.
```

```
11. capital-gain: continuous.
12. capital-loss: continuous.
13. hours-per-week: continuous.
14. native-country: United-States, Cambodia, England, Puerto-Rico, Canada, Germany,
Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras,
Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France,
Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala,
Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinadad&Tobago, Peru, Hong,
Holand-Netherlands.
```

Prediction task is to determine whether a person makes over 50K a year.

2 Decision Tree

2.1 ID3

ID3 (Iterative Dichotomiser 3) was developed in 1986 by Ross Quinlan. The algorithm creates a multiway tree, finding for each node (i.e. in a greedy manner) the categorical feature that will yield the largest information gain for categorical targets. Trees are grown to their maximum size and then a pruning step is usually applied to improve the ability of the tree to generalise to unseen data.

ID3 Algorithm:

1. Begins with the original set S as the root node.
2. Calculate the entropy of every attribute a of the data set S.
3. Partition the set S into subsets using the attribute for which the resulting entropy after splitting is minimized; or, equivalently, information gain is maximum.
4. Make a decision tree node containing that attribute.
5. Recur on subsets using remaining attributes.

Recursion on a subset may stop in one of these cases:

- every element in the subset belongs to the same class; in which case the node is turned into a leaf node and labelled with the class of the examples.
- there are no more attributes to be selected, but the examples still do not belong to the same class. In this case, the node is made a leaf node and labelled with the most common class of the examples in the subset.
- there are no examples in the subset, which happens when no example in the parent set was found to match a specific value of the selected attribute.

ID3 shortcomings:

- ID3 does not guarantee an optimal solution.
- ID3 can overfit the training data.
- ID3 is harder to use on continuous data.

Entropy:

Entropy $H(S)$ is a measure of the amount of uncertainty in the set S.

$$H(S) = \sum_{x \in X} -p(x) \log_2 p(x)$$

where

- S is the current dataset for which entropy is being calculated
- X is the set of classes in S
- $p(x)$ is the proportion of the number of elements in class x to the number of elements in set S.

Information gain:

Information gain $IG(A)$ is the measure of the difference in entropy from before to after the set S is split on an attribute A. In other words, how much uncertainty in S was reduced after splitting set S on attribute A.

$$IG(S, A) = H(S) - \sum_{t \in T} p(t) H(t) = H(S) - H(S \mid A)$$

where

- $H(S)$ is the entropy of set S
- T is the subsets created from splitting set S by attribute A such that $S = \cup_{t \in T} t$
- $p(t)$ is the proportion of the number of elements in t to the number of elements in set S
- $H(t)$ is the entropy of subset t.

2.2 C4.5 and CART

C4.5 is the successor to ID3 and removed the restriction that features must be categorical by dynamically defining a discrete attribute (based on numerical variables) that partitions the continuous attribute value into a discrete set of intervals. C4.5 converts the trained trees (i.e. the output of the ID3 algorithm) into sets of if-then rules. These accuracy of each rule is then evaluated to determine the order in which they should be applied. Pruning is done by removing a rule's precondition if the accuracy of the rule improves without it.

C5.0 is Quinlan's latest version release under a proprietary license. It uses less memory and builds smaller rulesets than C4.5 while being more accurate.

CART (Classification and Regression Trees) is very similar to C4.5, but it differs in that it supports numerical target variables (regression) and does not compute rule sets. CART constructs binary trees using the feature and threshold that yield the largest information gain at each node.

3 Tasks

- Given the training dataset `adult.data` and the testing dataset `adult.test`, please accomplish the prediction task to determine whether a person makes over 50K a year in `adult.test` by using ID3 (or C4.5, CART) algorithm (C++ or Python), and compute the accuracy.
- Your codes should output the decision tree. The decision tree can be represented with a nested dictionary structure or be depicted by using the `plotTree` function in the book `Machine Learning in Action`.
- Hints (You can refer to the book `Machine Learning` written by Zhou):
  1. You can process the continuous data with **bi-partition** method.
  2. You can use prepruning or postpruning to avoid the overfitting problem.
  3. You can assign probability weights to solve the missing attributes (data) problem.
- Please finish the experimental report named `E11_YourNumber.pdf`, and send it to `ai_2018@foxmail.com`

4 Codes

train.py

```python
from math import log
import pandas as pd
import numpy as np


def calEnt(dataSet):   # dataSet
    numEntries = len(dataSet)
    if numEntries == 0:
        return 0
    retDataSet = dataSet[dataSet['salary'] == ' >50K']
    retDataSet_ = dataSet[dataSet['salary'] == ' <=50K']
    Ent = 0.0
    value = len(retDataSet)
    value_ = len(retDataSet_)
    prob = float(value) / numEntries
    prob_ = float(value_) / numEntries
    if prob == 1 or prob_ == 1:
        return 0
    Ent = Ent - (prob * log(prob, 2) + prob_ * log(prob_, 2))
    return Ent


def spiltDataSet(dataSet, axis, value):  #
    retDataSet = dataSet[dataSet[axis] == value]
    del retDataSet[axis]
    return retDataSet


def spiltDataSet_con(dataSet, axis, min_value, max_value):  #
    retDataSet = dataSet[(dataSet[axis] >= min_value)
                         & (dataSet[axis] <= max_value)]
    del retDataSet[axis]
    return retDataSet

def isContinuous(attr):
    global continuousList
    return attr in continuousList

def calCon(dataSet, name):
    global baseEnt
    data = dataSet[name]
    data = data.sort_values()
    min_value = min(data)
    max_value = max(data) + 1
    num = len(data)
    counter = 0
```

```
bestFeature = -1
bestInfoGain = 0.0
while counter < num:
    if counter + 100 < num:
        counter += 100
        data_ = data[counter:counter+100]
        mean = data_.mean(axis=0) #
        subDataSet = spiltDataSet_con(
            dataSet, name, min_value, mean)
        subDataSet_ = spiltDataSet_con(
            dataSet, name, mean, max_value)
        prob = len(subDataSet)/float(num)
        prob_ = len(subDataSet_) / float(num)
        new_Ent = prob * calEnt(subDataSet) + \
                prob_ * calEnt(subDataSet_)
        infoGain = baseEnt - new_Ent
        if infoGain > bestInfoGain:
            bestInfoGain = infoGain
            bestFeature = mean
    else:
        data_ = data[counter:]
        mean = data_.mean(axis=0)  #
        subDataSet = spiltDataSet_con(
            dataSet, name, min_value, mean)
        subDataSet_ = spiltDataSet_con(
            dataSet, name, mean, max_value)
        prob = len(subDataSet)/float(num)
        prob_ = len(subDataSet_) / float(num)
        new_Ent = prob * calEnt(subDataSet) + \
                prob_ * calEnt(subDataSet_)
        infoGain = baseEnt - new_Ent
        if infoGain > bestInfoGain:
            bestInfoGain = infoGain
```

```
                bestFeature = mean
            break
    return [min_value, bestFeature, max_value]


def chooseBestSpilt(dataSet):
    global interval
    attr = dataSet.columns.values.tolist()
    num_attr = len(attr) - 1
    baseEnt = calEnt(dataSet)
    bestInfoGain = 0.0
    bestFeature = -1.0
    tol_num = len(dataSet) #
    for i in range(num_attr):#
        name = attr[i]
        if isContinuous(name):#
            cur_interval = interval[name]#
            interval_num = len(cur_interval)
            new_Ent = 0.0
            for x in range(interval_num - 1):#
                subDataSet = spiltDataSet_con(
                    dataSet, name, cur_interval[x], cur_interval[x+1])
                prob = len(subDataSet)/float(tol_num)
                new_Ent += prob * calEnt(subDataSet)
            infoGain = baseEnt - new_Ent
            if infoGain > bestInfoGain:
                bestInfoGain = infoGain
                bestFeature = i
        else:#
            value_list = dataSet[name].drop_duplicates()  #
            value_list = value_list.tolist()
            new_Ent = 0.0
            for value in value_list:
                subDataSet = spiltDataSet(dataSet, name, value)
                prob = len(subDataSet)/float (tol_num)
                new_Ent += prob * calEnt(subDataSet)
            infoGain = baseEnt - new_Ent
            if infoGain > bestInfoGain:
                bestInfoGain = infoGain
                bestFeature = i
    return attr[i] #

def majorCnt(classList):#
    classCount = {}
    for vote in classList:
        if vote not in classCount.keys():
            classCount[vote] = 0
        classCount[vote] += 1
    result = 0
    max_value = -1
    for key, value in classCount.items():
        if value > max_value:
            max_value = value
            result = key
    return result #

def createTree(dataSet):
    classList = dataSet['salary'].tolist()
    if classList.count(classList[0]) == len(classList):
        return classList[0]
    if len(dataSet.columns.values.tolist()) == 1:#
        return majorCnt(classList)
    bestFeat = chooseBestSpilt(dataSet)
    myTree = {bestFeat:{}}
    if isContinuous(bestFeat): #
        cur_interval = interval[bestFeat]  #
        interval_num = len(cur_interval)
        for x in range(interval_num - 1):
            min_value = str(cur_interval[x])
            max_value = str(cur_interval[x+1])
            interval_string = min_value + '-' + max_value
            subDataSet = spiltDataSet_con(
                dataSet, bestFeat, cur_interval[x], cur_interval[x+1])
            if len(subDataSet) != 0:
                myTree[bestFeat][interval_string]= createTree(subDataSet)
    else: #
        featValues = dataSet[bestFeat]
        uniqueVals = featValues.drop_duplicates()
        for value in uniqueVals:
            subDataSet = spiltDataSet(dataSet, bestFeat, value)
            myTree[bestFeat][value] = createTree(subDataSet)
    return myTree

def storeTree(inputTree, filename):
    import pickle
    fw = open(filename, 'wb')
    pickle.dump(inputTree, fw)
    fw.close()

attribute = ['age', 'workclass', 'fnlwgt', 'education',
             'education-num', 'marital-status', 'occupation',
             'relationship', 'race', 'sex', 'capital-gain',
             'capital-loss', 'hours-per-week', 'native-country',
             'salary']
continuousList = ['age', 'fnlwgt', 'education-num',
                  'capital-gain', 'capital-loss',
                  'hours-per-week']
dataSet = pd.read_csv('adult.data', names=attribute)
baseEnt = calEnt(dataSet)
```

```
interval = {}
for name in continuousList:
    interval[name] = calCon(dataSet, name)
myTree = createTree(dataSet)
storeTree(myTree, 'Tree.txt')
print('Finished')
```

test.py

```
import numpy as np
import pandas as pd

attribute = ['age', 'workclass', 'fnlwgt', 'education',
             'education-num', 'marital-status', 'occupation',
             'relationship', 'race', 'sex', 'capital-gain',
             'capital-loss', 'hours-per-week', 'native-country',
             'salary']
continuousList = ['age', 'fnlwgt', 'education-num',
                  'capital-gain', 'capital-loss',
                  'hours-per-week']



def isContinuous(attr):
    global continuousList
    return attr in continuousList

def classify(inputTree, featLabels, testVec):  #
    firstStr = list(inputTree)[0]
    secondDict = inputTree[firstStr]
    featIndex = featLabels.index(firstStr)
    classLabel = ''
    if isContinuous(firstStr):  #
        for key in secondDict.keys():
            interval = key.split('-')
```

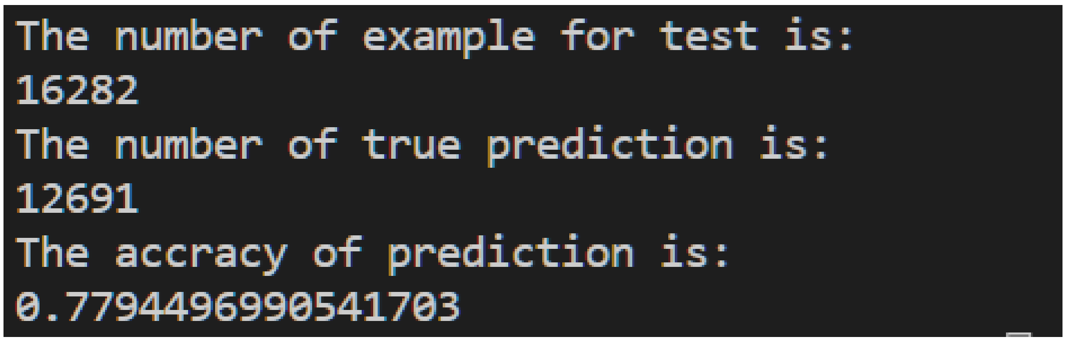
```
                min_value = float(interval[0])
                max_value = float(interval[1])
                value = float(testVec[featIndex])
                if value >= min_value and value < max_value:
                    if type(secondDict[key]).__name__ == 'dict':
                        classLabel = classify(
                            secondDict[key], featLabels, testVec)
                    else:
                        classLabel = secondDict[key]
    else:  #
        for key in secondDict.keys():
            if testVec[featIndex] == key:
                if type(secondDict[key]).__name__ == 'dict':
                    classLabel = classify(
                        secondDict[key], featLabels, testVec)
                else:
                    classLabel = secondDict[key]
    return classLabel

def grabTree(filename):
    import pickle
    fr = open(filename, 'rb')
    return pickle.load(fr)



myTree = grabTree('Tree.txt')
test_data = pd.read_csv('adult.test', names=attribute)
tol_test = len(test_data)
true_test = 0
for i in range(tol_test):
    testVec = test_data.iloc[i].tolist()
    truth = test_data.iloc[i]['salary']
    prediction = classify(myTree, attribute, testVec)
    prediction = prediction + '.'
    if prediction == truth:
        true_test += 1
print("The number of example for test is:")
print(tol_test)
print("The number of true prediction is:")
print(true_test)
print("The accracy of prediction is:")
print(true_test/tol_test)
```

5 Result



```
The number of example for test is:
16282
The number of true prediction is:
12691
The accracy of prediction is:
0.7794496990541703
```